VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2010 **Jiří Beran**

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Využití technologie Microsoft Software Factories pro zefektivnění vývoje modulárních spedičních a logistických aplikací pro operační systém MS Windows ve vývojovém prostředí MS Visual Studia 2008

Microsoft Software Factories utilization in MS Visual Studio 2008 for upgrading efectivity of transport and logistic systems

2010 **Jiří Beran**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Beran**

Studijní program: B2646 Informační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Využití technologie Microsoft Software Factories pro zefektivnění vývoje modulárních spedičních a logistických aplikací pro operační systém MS Windows ve vývojovém prostředí MS Visual Studia 2008

Microsoft Software Factories Utilization in MS Visual Studio 2008 for Upgrading Efectivity of Transport and Logistic Systems

Zásady pro vypracování:

Cílem práce je seznámení se s technologií Microsoft Software Factories, která umožňuje efektivní vývoj aplikací na platformě MS Windows. Hlavním úkolem tedy bude nastudovat a popsat možnosti pro zefektivnění vývoje modulárních a spedičních aplikací ve vývojovém prostředí MS Visual Studia 2008.
Předpokládá se splnění následujících bodů:
1. Popište možnosti efektivní tvorby aplikací budovaných na architektuře Software Factories.
2. Prostudujte použití architektury Smart Client Software Factories pro stavbu rozsáhlé modulární aplikace.
3. Získané poznatky následně aplikujte na návrh a tvorbu funkční ukázkové aplikace založené na architektuře Smart Client Software Factories.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Kocyan**

Datum zadání: 30.11.2008
Datum odevzdání: 07.05.2010

doc. Dr.Ing. Eduard Sojka
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. července 2010 ……

Poděkování

Chtěl bych touto cestou poděkovat vedoucímu bakalářské práce Ing. Tomáši Kocyanovi, za ochotu a lidský přístup. Dále bych chtěl poděkovat vedoucímu programátorovi ve firmě CID International, a.s. Ing. Petru Foltýnkovi za podporu, umožnění vzniku této práce a za vydatnou pomoc při řešení rozličných problémů.

Abstrakt

Tato práce se zabývá technologií Microsoft Software Factories, která umožňuje efektivní vývoj aplikací na platformě MS Windows. Hlavním úkolem tedy bylo popsat základní principy softwarových továren, popsat jejich konkrétní implementaci Smart Client Software Factories ve Visual Studiu 2008 a vytvořit ukázkovou aplikaci demonstrující základní práci s touto technologií.

Klíčová slova:

Smart Client Software Factories, Visual Studio, Microsoft, Composite UI Application Block, modulární aplikace.

Abstract

This work is dealing with the technology of Microsoft Software Factories which makes possible an efficient development of applications under MS Windows. The main task was to describe the basic philosophy of software factories and further to describe the concrete implementation of these applications under Visual Studio 2008 and create a demonstrative application which shows the basic work with this technology.

Keywords:

Smart Client Software Factories, Visual Studio, Microsoft, Composite UI Application Block, modular applications

Seznam použitých zkratek a symbolů

SCSF	Smart Client Software Factories
CAB	Composite UI Application Block
MVP	Model-View-Presenter
GAX	Guidance Automation Extension
GAT	Guidance Automation Toolkit
LINQ	Language Integrated Query

Obsah

1. Úvod

V současné době se klade velký důraz na efektivitu a kvalitu softwarových řešení. Praxe ale ukazuje, že projekty jsou drahé a často nedokončené v plánových termínech.

Tato práce má ukázat jednu z možných cest při vývoji aplikace, která by měla splnit požadavky zákazníka k jeho plné spokojenosti. Ukázaný směr vývoje se snaží udělat z programátora také vývojáře, po kterém bude jeho kód znovupoužitelný a přehledný.

Tato bakalářská práce by chtěla přispět k většímu prosazení softwarových továren v praxi.

2. Softwarové továrny

2.1. Co je softwarová továrna

Softwarová továrna je množina softwarových prostředků, které jsou nainstalovány ve vývojovém prostředí. Pomáhají vytvářet vysoce kvalitní aplikace. Při vývoji je ocení jak softwaroví architekti, tak vývojáři. Všechny softwarové továrny jsou určeny k tomu, aby pomáhaly tvořit aplikace, které mají konzistentní architekturu. Existují továrny pro různé typy aplikací. Jako mobilní, klientské aplikace či webové aplikace.

Základním smyslem softwarových továren je přivézt do výroby softwaru podobný způsob práce, jaký je obvyklý v ostatních průmyslových odvětvích (stavebnictví, automobilový průmysl, letectví atd.). Jde o znovupoužívání již jednou vyvinutých částí, které se poskládají v jeden funkční celek. Například v leteckém průmyslu je zcela běžné, že se na výrobě letadla podílí celá řada odvětví. Jedno odvětví se tak nestará o výrobu motoru a zároveň o výrobu sedaček pro cestující. Celý výrobní proces je tak rozložen na několik různých míst, čímž dochází k efektivnější dělbě práce. Každé odvětví se může více specializovat na svoji část a tím ji provézt efektivněji.

V softwarové výrobě se ale zatím příliš nesetkáváme se znovupoužíváním již hotových věcí. Je velice složité využívat již hotové části ve vlastní firmě, natož pak využívat externí komponenty, kde se navíc přidává i jistá míra nedůvěry.

V minulosti se již několik projektů snažící se vyřešit tento problém zrodilo, ale nebyly příliš úspěšné. Jako příklad můžeme uvést *COM* model, který zavedl možnost komponentového programování. Jako další příklad můžeme uvést *MDA* (Model Driven Architecture).

Softwarovým továrnám se daří tuto situaci pomalu měnit. Vize softwarových továren je taková, že vývojář bude spravovat softwarovou továrnu, s jejíž pomocí vytvoří výsledný produkt.

Je na místě ještě připomenout, že myšlenka softwarové továrny je již asi 15 let stará. (viz nástroje *GME*, nebo *MetaEdit*). Obecně se má za to, že tuto myšlenku pomohl více rozšířit až Microsoft, který dokázal koncept softwarové továrny integrovat do svého nástroje Visual Studio a dostal jej tak do produkčního prostředí bez podpory třetích stran.

2.2. Složení softwarové továrny

Obecná továrna obsahuje:

1. Softwarová aktiva
   Jedná se o dokumentaci, znovuaplikovatelné části kódu, referenční implementace, atd.
2. Softwarové Nástroje
   Do softwarových nástrojů můžeme zařadit různé generátory kódu, visuální designéry, průvodce atd.

Typické továrny poskytují šablony, nebo podobné nástroje, které mohou zásadním způsobem urychlit start vývoje nové aplikace. Napomáhá také vedoucím týmů automatizovat aktivity týkajících se životního cyklu aplikace.

Softwaroví architekti a vývojáři mohou nastavovat, upravovat nebo rozšiřovat projekt podle aktuálních potřeb. Toto můžeme zařadit mezi základní a klíčové vlastnosti softwarových továren.

Typický vývoj aplikace vypadá tak, že architekt aplikace nastaví softwarovou továrnu, provede potřebné úpravy a poté ji distribuuje vývojovému týmu. To v podstatě znamená, že jednotlivé části vývojového týmu mohou pracovat nezávisle na sobě. Většinou se tedy jedná o modulární aplikace.

2.3. Vývoj za pomocí softwarové továrny

Softwarové továrny zásadně potlačují problém tradičního vývoje aplikace, kdy jsou aplikace vyvíjeny bez většího využití znalostí získaných při vývoji podobných aplikací v minulosti. Softwarové továrny přistupují k tomuto problému způsobem vývoje prokázaného v praxi. Jsou tak snadněji přijímány vývojovými týmy.

Ve srovnání s konvenčním vývojovým procesem má softwarová továrna následující výhody:

Konzistence. Softwarová továrna využívá produktovou řadu. Ta představuje sadu aplikací, které sdílejí jak architekturu, tak vlastnosti. Jednotlivé instance jsou tedy konzistencí, což nám přináší značné snížení nákladů na údržbu a zaškolení uživatelů.

Kvalita. Jasně stanovený způsob práce (architektura). To znamená, že vývojáři věnují méně času psaní standardního kódu a věnují se více vytvářením vlastností, které jsou pro danou

aplikaci jedinečné. To redukuje pravděpodobnost aplikační či designové chyby. Aplikace vyvinuté s pomocí softwarové továrny lze před nasazením do provozu otestovat pomocí různých nástrojů v podobě Unit testů.

Produktivita. Znovupoužívání softwarových aktiv, možnost generování kódu či možnost vytvořit automatizované návody poskytuje více abstraktní pohled na vytvářený systém. Používání softwarových továren automatizuje a usměrňuje předepsaný vývoj následujícím způsobem.

- znovupoužití softwarových aktivit, zejména rozšiřitelné základní architektury, aplikačního rámce, a aplikačních bloků.
- poskytuje uvedení do kontextu a automatizované vedení
- generuje kód z modelů, které představují abstrakce z aplikačních prvků a mechanismů

Tyto postupy pomáhají zjednodušit, automatizovat, či dokonce eliminovat mnoho rutinních úkolů. Při použití softwarové továrny se tak snižuje časová náročnost projektu a tím i náklady na vývoj celého systému.

Co je tedy přesně zahrnuto v softwarové továrně?

Podobně jako aplikace, každá softwarová továrna je jedinečná a není tedy možné její složení paušalizovat. Nicméně většina softwarových továren postavených na Visual Studiu obsahuje následující komponenty:

Tovární schéma - ilustruje aktiva použitelná pro danou továrnu.

Referenční implementace - demonstruje příklad hotové aplikace, kterou lze vytvořit.

Pokyny a vzory - pomáhají vysvětlit rozhodnutí, která byla v rámci továrny udělána a vytváří motivaci pro vlastní rozhodnutí.

Postupy - popisují jak udělat konkrétní „věc“.

Recepty - recept jak zautomatizovat vybraný postup. Nástroj pro eliminaci rutinních činností.

Šablony - standardní prvky aplikace s vyznačenými místy pro konkrétní parametry pro možnost modifikace. Např. šablona pro výchozí nastavení *solution* Microsoft Visual Studia 2008**.**

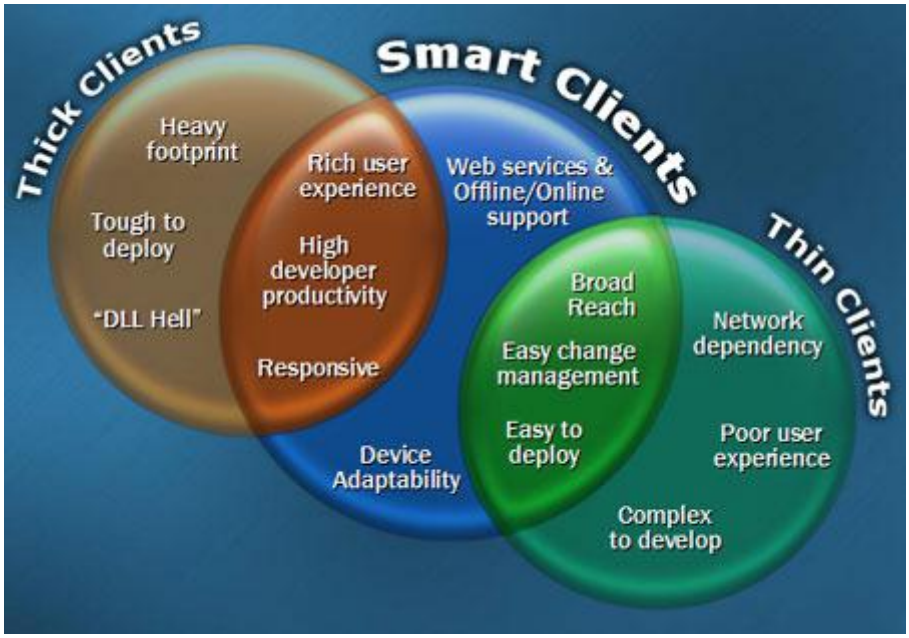
Designéry - umožňují visuální pohled na vytvářenou aplikaci (nebo její část). Příkladem nám může být GUI designér nebo např. diagram tříd.

Znovupoužitelný kód – je reprezentován komponentami. Integrace komponent v softwarové továrně nám snižuje množství ručně psaného kódu a umožňuje opětovné použití ve všech aplikacích.

3. Konkrétní implementace softwarové továrny

3.1. Smart Client Software Factory (SCSF)

Smart Client Software Factory je softwarová továrna, která slouží k vývoji tzv. chytrých klientů. Pojem chytrý klient zde chápeme jako běžnou desktopovou aplikaci, která se snaží využít a nabídnout možnost snadného nasazení, správu verzí, možnost spolupráce s webovými službami atd. Koncept chytrého klienta je zobrazen na obrázku 1.



Obrázek 1: koncept SCSF, zdroj Dokumentace SCSF

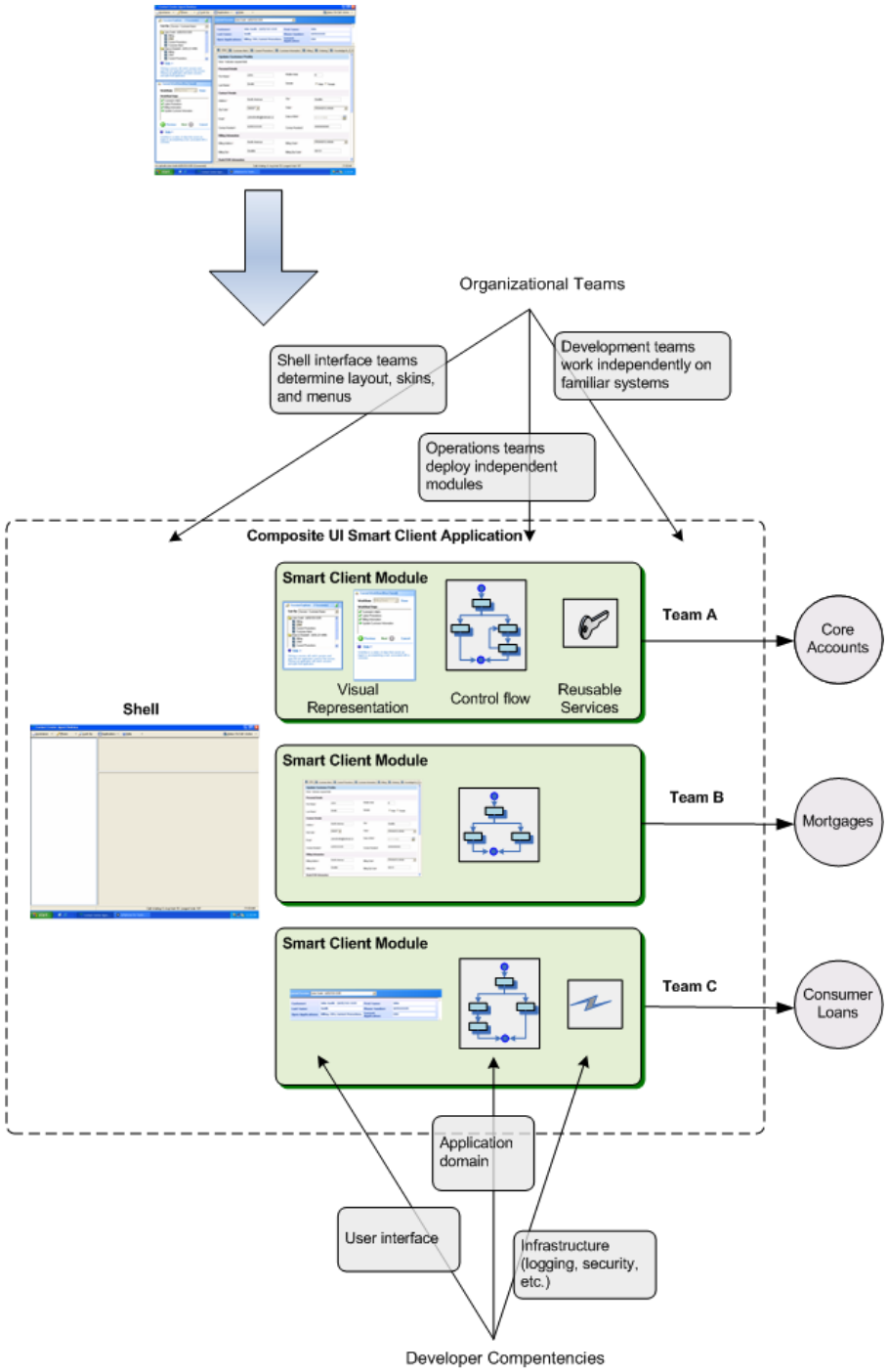
Továrna je založena na *Composite UI Application Blocku (dále jen CAB).* Jde o aplikační blok vytvořený skupinou *Microsoft patterns & practices*. Ta se věnuje návrhovým vzorům a vydává doporučení pro vývoj aplikací, efektivním praktikám atd. Doporučení jsou vydávána právě pomocí aplikačních bloků. Asi nejznámějším produktem této skupiny je *Enterprise Library*, *Composite WPF*, *Unity Application Block* a právě *Composite UI Application Block.*

3.1.1. Význam a popis klíčovných komponent používaných v Composite UI Application Blocku

Aplikace, která je vytvořená za pomocí *CAB* má tento profil:

- Aplikace je vytvořená z nezávislých, ale spolupracujících modulů.
- Při vývoji byli vývojáři modulů a jádra systému odděleni.
- Vývoj neprobíhá oproti vlastnímu vývoje od nuly. Vývoj také probíhá konzistentně.

Základní charakteristikou *CAB* je oddělení základu (*shellu*) aplikace od jednotlivých částí. Moduly tak lze vyvíjet více týmy současně a až při jejich společném běhu dojde k žádané kooperaci. Koncept této technologie demonstruje obrázek 2.



Obrázek 2: CAB, zdroj Dokumentace SCSF

Následující seznam vysvětluje hlavní pojmy, které jsou nutné pro pochopení *Composite UI Application Blocku.*

WorkItem

Dokumentace společnosti Microsoft definuje *WorkItem* jako run-time kontejner komponent, které spolupracují za účelem splnění *use case*. V mnoha případech je nejlepší přemýšlet o *WorkItems* jako o třídách, které obsahují kolekci dalších logicky propojených tříd.

Z pohledu kódu je pojem *WorkItem* pouze objektem ve třídě *WorkItem* v knihovně Microsoft.Patterns.CompositeUI. Každý *WorkItem* má kolekci objektů k němu přidružených, kterým je také povoleno obsahovat další objekty.

Typy *WorkItem* kolekcí jsou následujicí:

1. Kolekce objektu, které mohou obsahovat téměř cokoliv, protože se jedná o kolekci typu *System.Objects*
2. Kolekce CAB funkcí (*CAB Services*)
3. *WorkItem* kolekce, která obsahuje stromovou strukturu dceřiných objektů typu *WorkItem*

Services

CAB dokumentace definuje *Services* (služby) jako podpůrnou třídu, která poskytuje funkcionalitu jiným komponentám. Jinými slovy služby umožňují snadný přístup k částem funkcí, které mohou být používány často v celé aplikaci. Autentifikace nebo autorizace webové služby jsou dobrými příklady funkcí, které mohou být uvedeny jako služba.

Module

Obvykle se jedná o samostatnou dll knihovnu, která může být generována on-demand a má v sobě ucelenou funkčnost. *Module* obsahuje kolekci *WorkItemů* a protože obecně jeden *WorkItem* náleží jednomu případu použití (*Use case*) lze říci, že module je balíček obsahující související případy použití.

Moduly nejsou přidány do *Shell* pomocí klasických projektových referencí, ale jsou nahrány pomocí souboru *ProfileCatalog.xml*. Z toho plyne, že při přidání dalšího modulu není nutná rekompilace celé aplikace.

ProfileCatalog

Jak již bylo řečeno, moduly nejsou přidány do shellu pomocí klasických referencí, ale jsou svázány tenkou vazbou při jejich spuštění. Jeden z principů využívá *ProfileCatalog*. Pokud jej ale nechcete používat, můžete tuto funkčnost přepsat vlastní implementací.

Jedná se o nastavení, které specifikuje, které jednotky a služby musí být načteny do aplikace. Ve výchozím nastavení se jedná o nějaký XML soubor, který je umístěn v adresáři aplikace. Můžeme vytvořit a zaregistrovat třídu *IModuleEnumerator*, pomocí které lze výchozí nastavení změnit. Budeme tak moci načítat nastavení z jiného umístění, například z webové služby.

ModulLoader

Jedná se o třídu, která se stará o načtení všech modulů specifikovaných v *ProfileCatalog*. V podstatě si přečte požadovanou konfiguraci v *ProfileCatalog* a dynamicky načte určenou *assembly*. Následně vyhledá třídu *ModuleInit* uvnitř této *assembly*.

ModuleInit

Každý modul se skládá z třídy *ModuleInit*, která se stará o načtení všech potřebných služeb a *WorkItems* z modulu. Třída úzce souvisí s *UIExtensionSite* (rozebrána níže).

Shell application

Jádro aplikace, které je zodpovědné za inicializaci aplikace, dynamicky zaveditelné moduly dle dané konfigurace a spuštění sady základních služeb pro *smart client* aplikaci.

Shell

Představuje hlavní formulář aplikace. Vytváří uživatelské rozhraní aplikace, které je společné pro všechny zaveditelné moduly. Vždy hostí základní (kořenový) *WorkItem*, který je vstupním bodem do jiných částí aplikace. Tedy do služeb, modulů a *WorkItems*, které byly vytvořeny a zaregistrovány pod těmito *WorkItems*.

WorkSpace

Jedná se o kontejner, který se stará o vlastnictví a zobrazení uživatelského rozhraní vytvořeného *WorkItem*. Nejčastěji je přidán k *Shell* a působí jako kontejner pro části uživatelského rozhraní. Ty by měly být dynamicky naplněny položkami, které jsou poskytovány od *WorkItems*. Každý *UserControl* může být *workspace* pomocí zavedení rozhraní *IWorkSpace*. *CAB* zahrnuje standardní *container controls* jako *workspace*.

UIExtensionSite

Jde o zvláštní místo pro rozšířené části *Shell* jako například menu. Oproti *WorkSpace* předpokládá, že bude použito pro části uživatelského rozhraní, které nebudou přepsány od

WorkItems. Předpokládá se pouze jejich rozšíření. Například přidání nové položky do menu a podobně.

Command

Jedná se o základní třídu *CAB* pro zavedení *Command* vzoru. Jsou podporovány běžné příkazy vytvořené ručně nebo deklarativní příkazy použitím *[CommandHadler]* jako atributu k metodě. Tyto metody se pak chovají jako obsluha těchto událostí.

EventPublication

Používá se pro volné spojení událostí. Jedná se o standardní události .NET Frameworku, které jsou navíc označeny atributem *[EventPublication]*. Události jsou jednoznačně identifikovány pomocí jedinečného řetězce *URI*. Jedině *subscribers* mohou používat *subscriptons* se stejnou událostí *URI*. *Subscribers* potřebuje mít metodu se stejným popisem jako událost užívaná *[EventPublication]* a zároveň musí být označena atributem *[EventSubscription]*. Standardní událost .NET Frameworku se nepoužívá, protože jednotlivé moduly na sebe nemají reference.

EventSubscription

Je protistranou *EventPublication*. Jakákoliv třída, která potřebuje *subscribe* událost s konkrétní událostí, musí implementovat *event handler*, který odpovídá metodě *EventPublication*. Tohoto správce události musíme označit atributem [*EventSubscription*] a *CAB* se vždy ujistí, že *subscriber* bude vyvolán vždy, když *publisher* zavolá danou událost.

Model

Bussines entita *WorkItem* procesů.

View

Standardní prvek .NET Frameworku. Je zodpovědný za visuální prezentaci modelu (nebo jeho části) uživateli. Umožňuje modifikaci obsahu za pomocí uživatelských ovládacích prvků. *View* by mělo implementovat pouze logiku pro obsluhu uživatelského rozhraní. Běžná aplikační logika by měla být implementována v *presenteru*.

SmarPart

V .NET Frameworku jde o standardní user control s přidaným atributem [SmartPart]. Může navíc implementovat rozhraní *ISmartPartInfoProvider*. *SmartPart* o sobě poskytuje informace. Například titulek a popis.

Controller

Třída, která implementuje logiku k modifikaci modelu. Jeden *controller* může modifikovat *model* representovaný více *view*.

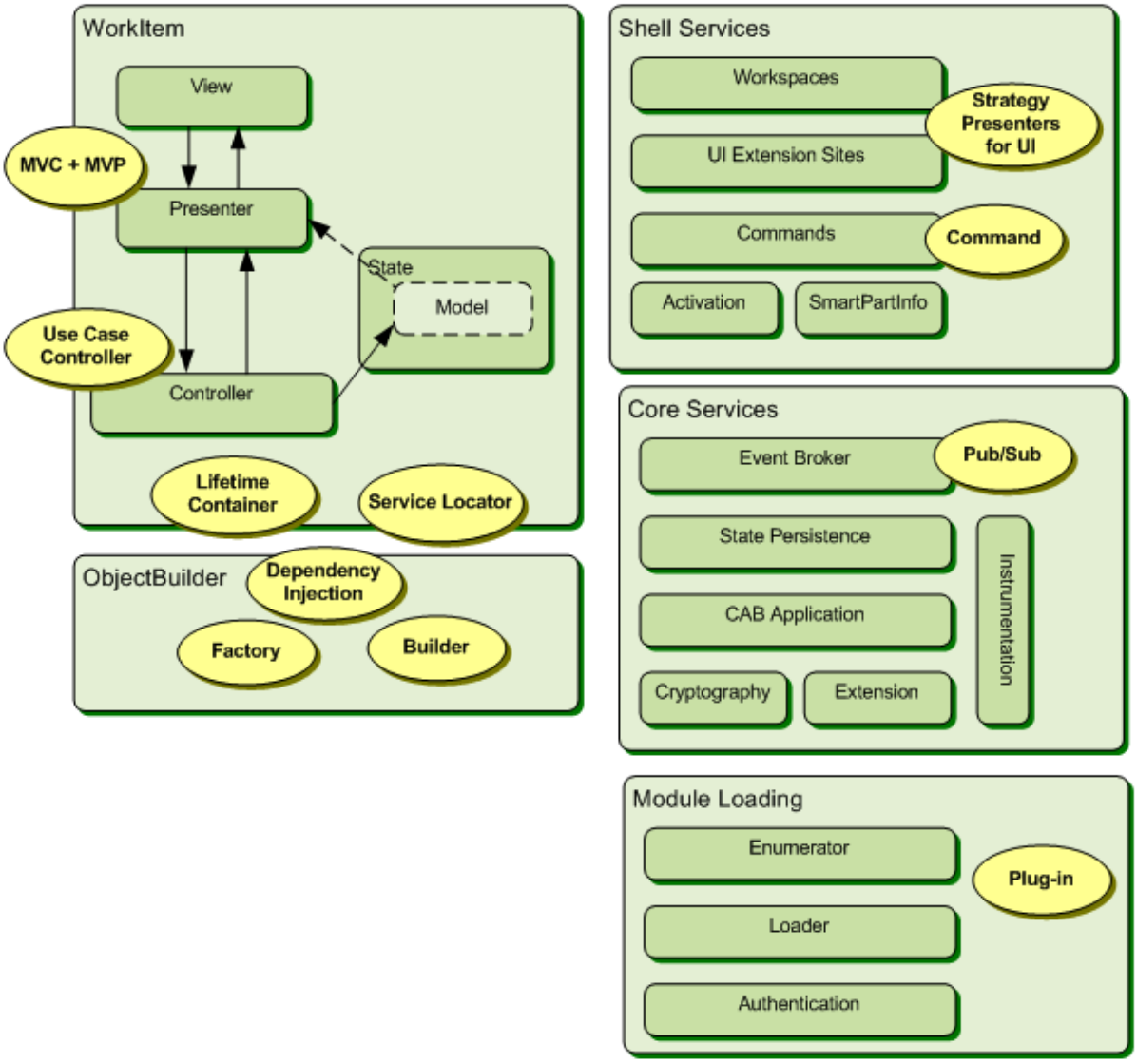
ObjectBuilder

Komponenta, která vytváří továrnu pro vytvoření objektů, které požadují specifickou strategii k tomu, aby byly vytvořeny, nebo které potřebují rysy jako automatická konkretizace a inicializace závislých objektů při vytváření instancí. *ObjectBuilder* kombinuje úlohy jako továrna, *dependency injection* a strategie frameworku.

Dependency Injection

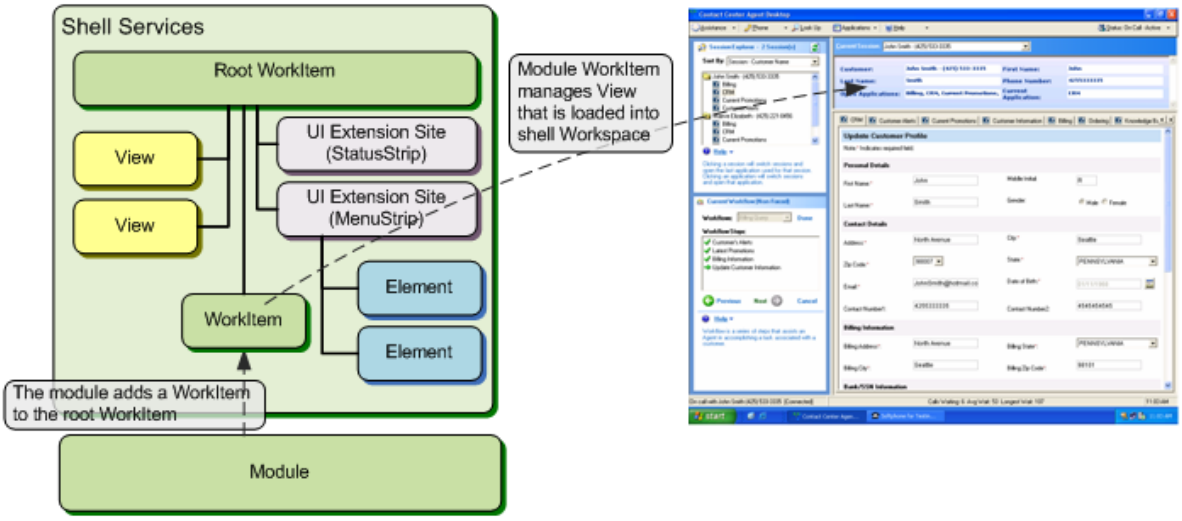
Jedná se o jeden z návrhových vzorů, který umožňuje továrně automaticky vytvořit a inicializovat vlastnosti a členy objektů i s jejich závislostmi. V podstatě umožňuje jedné třídě použít jinou třídu bez jakéhokoliv odkazu na ni. Tuto funkčnost zprostředkuje *ObjectBuilder*.

3.1.1.1. *Využití návrhových vzorů v CAB*



Obrázek 3: Návrhové vzory v CAB, zdroj Dokumentace SCSF

Celá architektura využívá několik návrhových vzorů, například vzor *Model-View-Presenter* (MVP), *Dependency Injection*, *Command*, *Plugin* a jiné. Jednotlivé komponenty, jsou právě kvůli použitým návrhovým vzorům volně svázány a například implementace *Dependency Injection* je tak obsáhlá, že pro ni byl vytvořen samostatný projekt *Object Builder*. Jednotlivé komponenty, v případě že jejich funkčnost z nějakého důvodu nevyhovuje, lze nahradit vlastní implementací právě díky jejich volnému svázání.



Obrázek 4: Aplikace pomocí CAB, zdroj Dokumentace SCSF

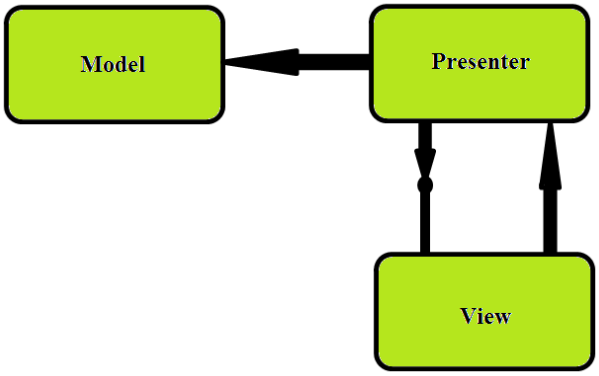
Na obrázku č.4 je vidět struktura aplikace vytvořené pomocí *CAB*. *Shell* obsahuje *WorkItem* a poskytuje modulům *UI extensit sites* – části uživatelského rozhraní, které jednotlivé moduly mohou měnit. Každý modul může přidat svůj vlastní *WorkItem* a poskytnout *SmartPart* (pohled), který může vyvolat vlastní okno aplikace. Na obrázku č.4 vpravo je vidět možný příklad uživatelského rozhraní, které se skládá z několika částí vytvořených jednotlivými moduly. Rozhraní je jako celek synchronizováno. Pokud jsou tedy v jednom modulu editovány informace o jednom produktu, všechny ostatní pohledy (moduly) se mu věnují také. Tímto je tedy splněna jedna ze základních myšlenek, že moduly jsou oddělené, ale přesto spolu spolupracují.

3.1.2. Návrhový vzor Model-View-Presenter (MVP)

Smart Client aplikace obsahují různé ovládací prvky, které jsou ovládány událostmi, a různou logiku, která reaguje na podněty uživatele. Psaní aplikační logiky ve stejné třídě by bylo velice nepřehledné a obtížné v mnoho ohledech. Velice by to znesnadňovalo testování, týmovou práci a podobně. Navíc by bylo obtížné sdílet kód pro jiná uživatelská rozhraní, která požadují stejnou obsluhu událostí.

Tento problém úspěšně řeší návrhový vzor Model-View-Presenter.

Definice:



Obrázek 5: MVP, zdroj Dokumentace SCSF

MVP může být chápán spíše jako softwarová architektura než jako návrhový vzor. Rozděluje aplikaci do tří vrstev. Na datový model, uživatelské rozhraní a řídící logiku. Základní myšlenkou je, že modifikace jedné vrstvy má pouze minimální vliv na vrstvy ostatní.

Základní vlastnosti:

Model neví, že existuje *view* a *presenter*.

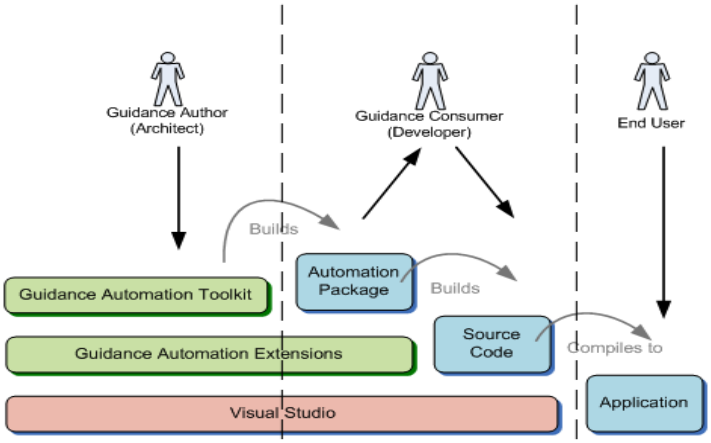
View má v sobě referenci na *presenter* a předává mu uživatelské události. Ve *view* tedy není implementována vůbec žádná *bussines* logika aplikace.

Presenter neobsahuje přímou referenci na *view*, ale pouze obsahuje referenci na jeho *rozhraní IView*, přes které s ním komunikuje. To nám dává možnost jednoduché výměny *View* za jiné. *Presenter* tedy „seznámí“ *view* s modelem (nikdy ne naopak) a zpracovává uživatelské pokyny.

Model, v ideálním případě, obhospodařuje pouze práci se získáváním dat z databáze. Ve většině případů model tvoří více tříd, ale z pohledu architektury jde stále o jeden celek. Model udržuje aplikační data a aplikační entity.

3.1.3. Guidance Automation Extension (GAX) a Guidance Automation Toolkit (GAT)

Gauidance Automation Extension je běhové prostředí pro balíčky návodů, zatímco Guidance *Automation Toolkit* je nástroj určený především pro softwarové architekty, kterým umožňuje tyto balíčky vytvářet. Vztahy mezi těmito nástroji demonstruje obrázek č.6.



Obrázek 6: Vztah GAX a GAT, zdroj Dokumentace SCSF

Balíčky vytvořené pomocí GAT obsahují následující komponenty:

Recepty

Automatizují aktivity, které by vývojáři museli provádět ručně. Tím se eliminuje možnost lidské chyby a navíc se šetří práce. Danou činnost lze aplikovat na jeden nebo i více elementů jednoho sestavení.

Akce

Jedná se o automatickou jednotku receptu. Recept se skládá z jednotlivých akcí.

Šablony pro transformaci textových šablon

Jedná se o šablony, které jsou tvořeny textem a skriptlety.

Průvodci

Sbírají od vývojáře informace, které následně předají receptu.

Typové konvertory

Převádějí pole z uživatelského rozhraní do typové formy.

Šablony pro Visual Studio

Mohou přidávat projekty do stávající *solution* nebo vytvářet úplně nové. Jsou psány v XML.

3.1.4. Instalace SCSF pro Visual Studio 2008 SP1

Instalace SCSF předpokládá nainstalovaný .NET Framework 3.5, *Guidance Automation Extensions* a *SQL Server 2005 Compact Edition*. Volitelně pak *Enterprise Library* 3.1 a *Guidance Automation Toolkit*.

Instalace pro Visual Studio 2008 bez SP1 a Visual Studio 2010 je bezproblémová, kvůli větším změnám s příchodem opravného balíku SP1 ale musíme po spuštění instalačního balíku sáhnout k úpravám určitých souborů, protože oficiální instalační balíček pro SP1 neexistuje.

Nebudu zde zabíhat do podrobností proč to ve SP1 nefunguje, protože by této práci nic nepřinesly.

Pro upravení SCSF budeme potřebovat upravit zdrojový kód *Guidance Package*.

Otevřeme tedy GuidancePackage.sln a najdeme soubor *ViewTemplateReferenceVB.cs* a *ViewTemplateReferenceCS.cs* ve složce *references* ve *SmartClientFactoryPackage* projektu.

Pro oba soubory nalezneme řádek:

```
Line # 154: if (reference.Identity == referenceIdentity) return true;
```

a nahradíme jej:

```
Line # 154: if (reference.Name == referenceIdentity) return true;
```

Dále nalezneme *CreateSmartClientFactoryBusinessModuleCommon.xml* a *CreateSmartClientFactoryFoundationalModuleCommon.xml* ve složce *Recipes\Common*.

A opět v obou souborech nalezneme řádek

```
Line #104: <Input Name="ModuleName" RecipeArgument="ModuleName"/>
```

a nahradíme jej:

```
Line #104: <Input Name="ModuleName" RecipeArgument="ModuleNamespace"/> />
```

Dále nalezneme *ProfileCatalog.xml* ve složkách *Templates\Solutions\Projects\Shell.Basis.VB*, *Templates\Solutions\Projects\Shell.Extended.VB* a *Templates\Solutions\Projects\Shell.Extended.CS*

Nalezneme řádky:

```
Line #4:  <ModuleInfo AssemblyFile="Infrastructure.Layout.dll" />
Line #12: <ModuleInfo AssemblyFile="Infrastructure.Module.dll" />
```

a nahradíme je:

```
Line #4:  <ModuleInfo
AssemblyFile="$RootNamespace$.Infrastructure.Layout.dll" />
Line #12: <ModuleInfo
AssemblyFile="$RootNamespace$.Infrastructure.Module.dll" />
```

Nakonec *Guidance Package* zkompilujeme a zaregistrujeme.

3.1.5. Obsah SCSF

Po nainstalování se vytvoří v operačním systému několik odkazů, které vedou na soubory s dokumentací. Dále jsou připraveny dvě ukázkové implementace. *AppraiserWorkbench* a *BankBranchWorkbench*.

Dokumentace neobsahuje pouze popis architektury, ale poskytuje i několik návodů jak co udělat.

Struktura dokumentace:

- Úvod do SCSF
  - Jak rychle začít
  - Instalace
  - O autorech, copyright
- Vysvětlení základních konceptů
  - Co jsou softwarové továrny
  - Koncept chytrých klientů

    - Popis Composite UI Application Blocku
    - Popis principu automatizovaných návodů
- Vývoj s SCSF
    - Vysvětlení použitých návrhových vzorů
    - How-to témata – jsou zde uvedeny jednotlivé vývojářské úlohy popsané krok za krokem.
    - Ostatní témata
- Modifikace SCSF
    - Modifikace automatizovaných návodů
    - Modifikace dokumentace

Mimo dokumentace se spolu s SCSF se do Visual Studia nainstaluje několik *guidance packages*:

- **Smart Client Application**
  Jde o šablonu, pomocí které se vyvoří standardní solution.

- **Add Business Module**
  Šablona, pomocí které se vytvoří projekt obsahující WorkItem

- **Add Foundation Module**
  Šablona, pomocí které se vytvoří projekt, který poskytuje služby jiným modulům.

- **Add View (with presenter)**
  Recept, který vytvoří View(pohled) a presenter dle návrhového vzoru MVP

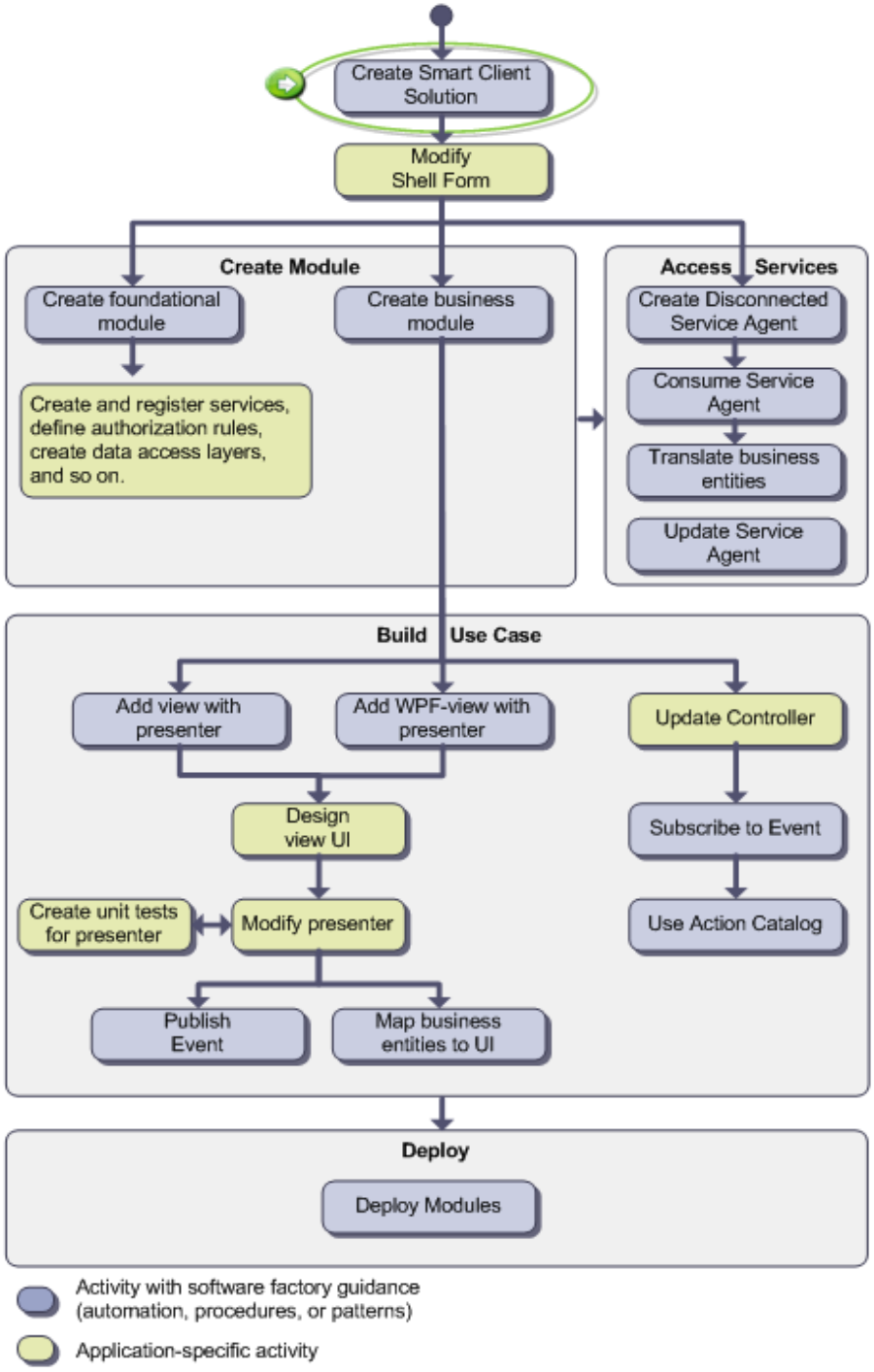
- **Add Smart Web Reference**
  Recept, který vytvoří třídy a příkazy pro volání různých webových služeb podle vzoru *Asynchronous Web Service Invocation with Timeout*

- **Add Event Publication**
  Recept, který do již existující třídy přidá kód, který deklaruje a publikuje události.

- **Add Event Subscription**
  Recept, který do již existující třídy přidá metodu, která se přihlásí k odběru požadované události.

3.1.6. Popis práce s SCSF

Schematické znázornění postupu při vytváření *smart klient* aplikace je znázorněno na obrázku č. 7. Základní kroky budou popsány v následujících částech této práce.

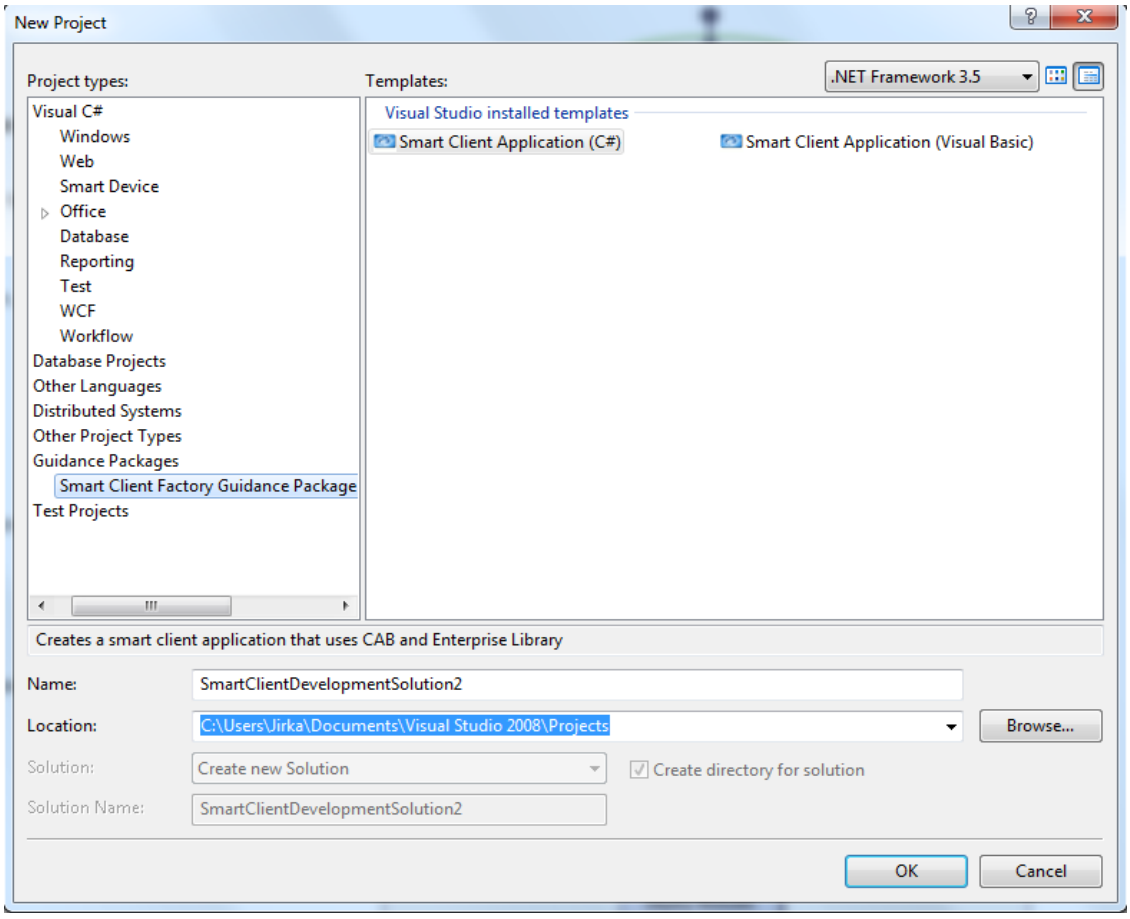


Obrázek 7: SCSF, zdroj Dokumentace SCSF

3.1.6.1. Vytvoření sestavení

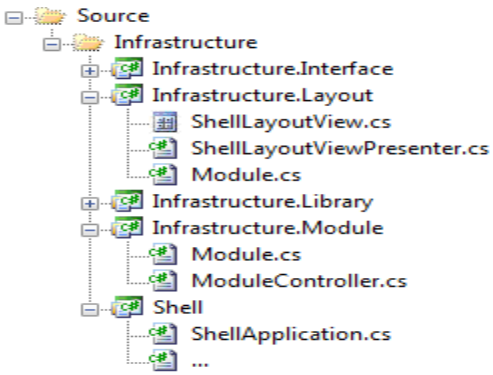
Po instalaci se ve Visual Studiu vytvoří nový typ projektu *Smart Client Factory Guidance Package*. Je k dispozici verze jak pro C#, tak pro Visual Basic.

Pro vytvoření nového projektu za pomoci Smart Client Software Factory musíme podstoupit tyto kroky:



Obrázek 8: Dialog pro vytvoření nového projektu

1. Otevřít Visual Studio, vybrat File – New – Projekt a vybrat *Smart Client Factory Guidance Package*
2. V prvním kroku průvodce je potřeba zadat tyto informace:
    - Umístění sestavení Enterprise Library.
    - Kořenový namespace pro chytrého klienta. Tento namespace se automaticky použije ve všech modulech přidaných do solution.
3. Zmáčknout Finish tlačítko a SCSF následně vytvoří několik projektů.



Obrázek 9: Vygenerované projekty SCSF

Smart Client Software Factory přidá řadu základních služeb, které jsou poskytované *CAB*. Tyto základní služby jsou zahrnuty v samostatných projektech vytvořených v SCSF. Následující tabulka č.1 popisuje jednotlivé projekty, které byly vytvořeny při dokončení průvodce.

Projekt	Popis
Infrastructure.Interface	Definuje rozhraní pro základní služby vytvořené *SCSF*. Přidané služby jsou uvedeny v tabulce č.2.
Infrastructure.Library	Obsahuje implementaci dodatečných služeb vytvořených pomocí *SCSF*.
Infrastructure.Module	Prázdný modul, do kterého si můžete přidat vlastní infrastructuru služeb. Ve výchozím nastavení se tento modul skládá z prázdného *modul controlleru* (což je výchozí kořenový *WorkItem* pro modul). V tomto modulu musíte pouze přidat služby, které používá samostatná aplikace. To znamená nezávislé služby na jednotlivých modulech.
Inftrastructure. Layout	Obsahuje *SmartPart* a *presenter* pro základní rozvržení aplikace.
Shell	Skutečná windows aplikace. Obsahuje konfiguraci a skutečný hlavní formulář, který je hostitelem základního rozvržení aplikace.

Tabulka 1: Projekty po dokončení úvodního průvoce

Následující tabulka č.2 popisuje služby, které se přidaly po založení projektu:

Služba	Popis
Entity Translator Base Class	Základní třídy, pro překlad entit. Používají se pro complexní *smart clienty*, kde je obvykle potřeba provést mapování ze smart klienta na webovou službu, pomocí volného spoje.
Basic SmartPart Info Classes	*CAB* umožnuje vytvářet třídy, které implementují ISmartPartInfo. Potřebujeme tedy tuto implementaci ve třídě, pokud chceme zobrazit další informace o *SmarPart*. Například, pokud chceme zobrazit titulek v záložce, musíme splnit imlementaci *ISmartPartInfo*. *SCSF* vytváří základní implementace tohoto rozhraní, které jsou často používány.
ActionCatalog Service	Umožňuje nám definovat konkrétní metody jako akce, které jsou automaticky aktivovány pokud to nastavení zabezpeční uživatele umožňuje.
DependentModuleLoader	Ve výchozím nastavení, *CAB* přichází s jednoduchým profilem katalogu a *module loaderu*. Jeho konfigurace je uložena v souboru *ProfileCatalog.xml* v adresáři aplikace. *SCSF* přidává nový *loader*, který umožnuje zadat vztahy mezi jednotlivými moduly. Pomocí *DependentModuleLoader* můžeme určit, zda jeden modul zavisí na jiném a aby případně byly načteny dříve než požadovaný modul.
WebServiceCatalog Service	Pomocí této služby můžeme načíst moduly, které mají být spuštěny při spuštění chytrého klienta z webové služby.

Tabulka 2: Služby při založení projektu

3.1.6.2. Vytvoření nového bussines modulu

Nejprve je potřeba vybrat z kontextové nabídky *Add Bussiness* module. Opět platí, že se vytvoří samostatný projekt, pro jeho rozhraní. Vytvoření samostatné knihovny pro interface je velmi důležité, protože nám umožnuje sdílet typy s ostatními moduly.

Včetně :

Rozhraní pro služby registrovaných na tento modul.

Argumentů událostí pro události vyvolaných tímto modulem.

Konstant pro jména příkazů a událostí, vyvolaných tímto modulem.

Tabulka číslo 3 popisuje součásti a jejich funkce, které byly vytvořeny když byl přidán nový *bussines modul*.

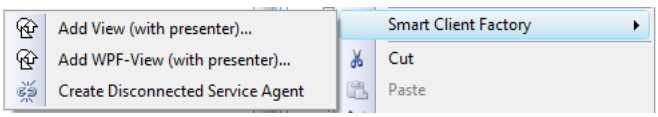
Část	Popis
Module třída	Implementace *ModuleInit* základní třídy pro *CAB*. Načítá všechny služby a první stupeň *WorkItems* modulu.
ModuleController	*SCSF* nevytváří přímo *WorkItems*, vytváří *WorkitemsControllery* které implementují skutečnou logiku. SCSF vytvoří kořenový *WorkitemController* pro každý bussines modul. *WorkItemController* je vstupním bodem pro všechny *sub-WorkItemy*(pokud existují). A ten je nazýván *ModuleController*
Constans folder	Obsahuje soubory pro definici konstant pro *UIExtensionSites*, *workspaců*, příkazů a událostí, které patří do tohoto modulu.

Tabulka 3: Součásti a funkce nově vytvořeného bussines modulu

Po přidání modulu můžeme začít vytvářet *WorkItems* a *SmartParts* potřebné pro imlementaci aplikační logiky prováděné v modulu.

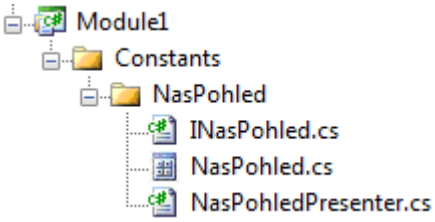
3.1.6.3. Vytvoření nového View s presenterem

Nyní musíme do modulu přidat samotný pohled. Otevřeme kontextové menu na *bussines modulu* a vybereme *Add* view (with presenter).



Obrázek 10: přidání pohledu

Po zadání jména pohledu nám recept vygeneruje nový pohled. Nyní s ním pracujeme stejně jako se standardním *user controlem*.



Obrázek 11: Vytvořený pohled

3.1.6.4. Konfigurace pohledu v jádru aplikace

Nyní nás čeká konfigurace zobrazení pohledu v *shellu*. (pro názornost v následujícím kódu předpokládáme pojmenování pohledu „*NasPohled*“ a použití jazyka C#).

V projektu *business modulu* nalezneme soubor *ModuleController.cs*.

Přidáme následující kód:

```
using HelloWorldApplication.Infrastructure.Interface.Constants;
```

V metodě AddViews přidáme:

```
// Prida NasPohled view (smart part) do WorkItemu a zobrazi jej v praven
workSpace
NasPohled hwview = ShowViewInWorkspace< NasPohled
>(WorkspaceNames.RightWorkspace);
```

Tím máme k dispozici zkompilovatelnou aplikaci, která obsahuje menu a *NasPohled*.

Nyní si ukážeme použitelnější aplikaci, která bude vyvolávat *NasPohled,* až po jeho vybrání z menu.

Zrušíme předchozí kód v metodě *AddView* a vytvoříme menu s novým pohledem.

V *CommandNames.cs* definujeme název pohledu:

```
public const string ZobrazNasPohled = "ZobrazNasPohled";
```

V ModulleController.cs přidáme do metody ExtendToolStrip kód, který rozšíří menu aplikace o náš modul:

```
AddToolStripButton(Constants.CommandNames.ZobrazNasPohled, "Náš pohled");
```

Dále musíme metodu *AddToolStripButton* naimplementovat.

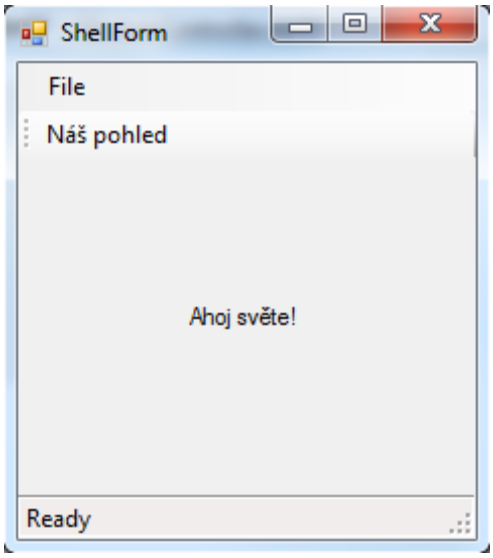
Například takto:

```
private void AddToolStripButton(string commandName, string text)
 {
    ToolStripButton button = new ToolStripButton();
    button.Text = text;
    button.ToolTipText = text;

    // pridani tlačítka do MainToolBar.

WorkItem.UIExtensionSites[UIExtensionSiteNames.MainToolbar].Add(button);

    // Associate the Click event of the button to a command
    WorkItem.Commands[commandName].AddInvoker(button, "Click");
 }
```

Dále vytvoříme *CommandHandler*, který bude zpracovávat pokyn pro zobrazení pohledu, například takto:

```
[CommandHandler(Constants.CommandNames.ZobrazNasPohled)]
 public void OnShowHelloWorldMessage(object sender, EventArgs e)
 {
// Prida NasPohled view (smart part) do WorkItemu a zobrazi jej v praven
workSpace
 HelloWorldView hwview =
ShowViewInWorkspace<ZobrazNasPohled>(WorkspaceNames.RightWorkspace);
 }
```

Nyní můžeme aplikaci zkompilovat a spustit. Náhled právě vytvořené aplikace vidíme na obrázku č.12.



Obrázek 12: Náhled aplikace

3.2. Ukázková aplikace v SCSF

V této kapitole postupně představím aplikaci, která demostruje základní možnosti technologie SCSF. Aplikace je reálně nasazená a funkční.

Účelem této práce není detailní popis ukázkové aplikace, proto pouze stručně popíši její účel.

Aplikace slouží pro administraci modulární aplikace Lori firmy CID International, a.s., která je také napsána za pomocí technologie SCSF.

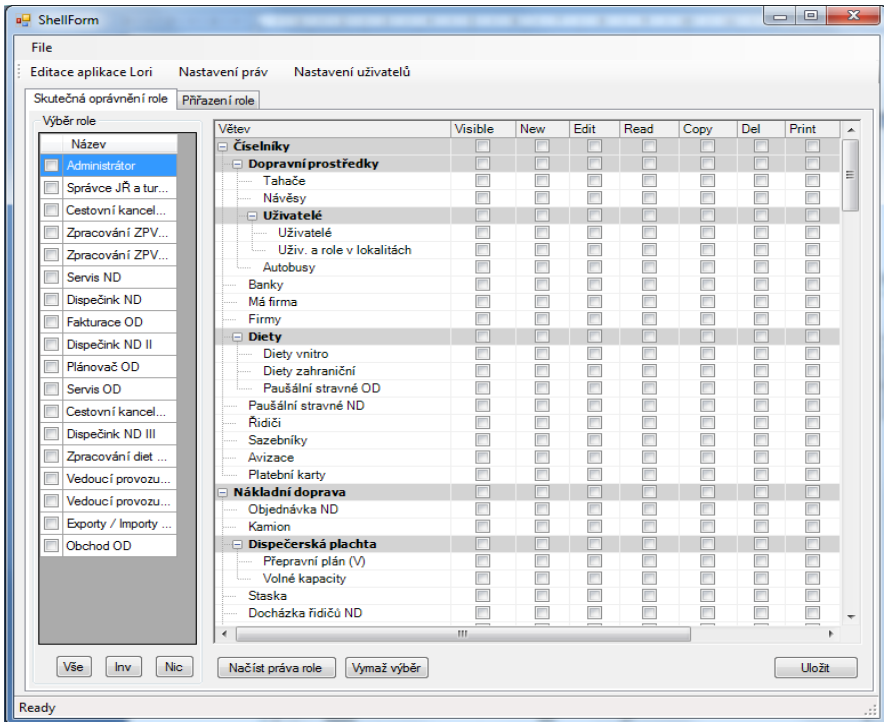
3.2.1. Popis jednotlivých bussines modulů:

3.2.1.1. Nastavení oprávnění

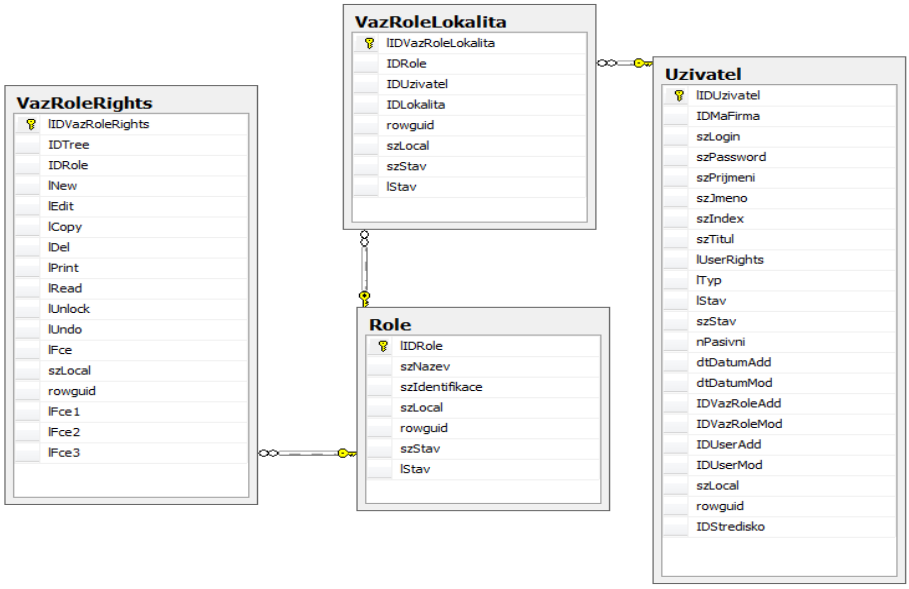
Modul je vytvořen tak, aby v levé části obsahoval statický seznam rolí s možností výběru jednotlivě i vícenásobně a nástroji pro usnadnění výběru. V pravé části pak jsou dvě záložky, přičemž první je určena k definici práv role a druhá pro přiřazení role uživateli a lokalitě.

V modulu je kontrolována návaznost práv, aby nedocházelo k situacím, kdy podvětev má větší práva než hlavní větev stromu. Pro pohodlnější obsluhu lze využít kontextové menu, které nabízí označení celého sloupce nebo řádku, dědění práv z nadřazeného uzlu a smazání veškerých práv.

Následující obrázky 13 a 14 ukazují náhled na dialog a strukturu databáze pro nastavení práv.



Obrázek 13: Nastavení práv

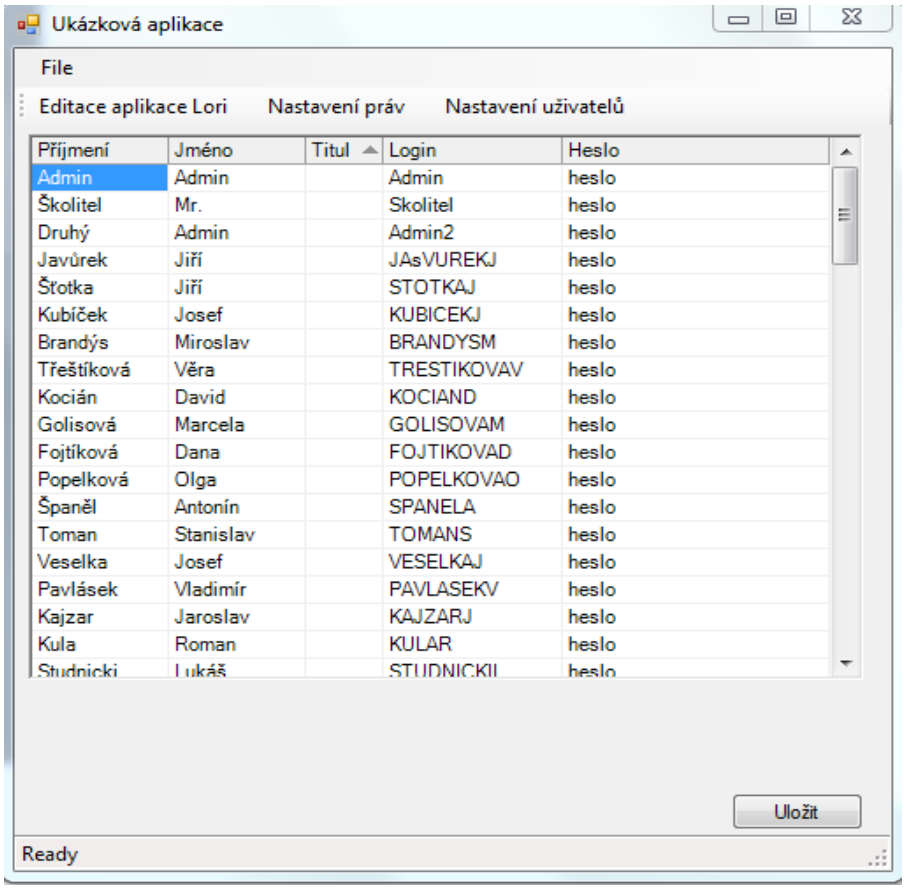


Obrázek 14: struktura části databáze, nastavení práv

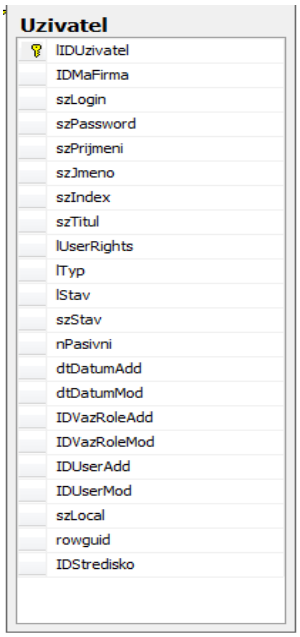
3.2.1.2. Nastavení uživatelů

Slouží pro vytváření, editaci a mazání uživatelů. Nastavení uživatelů se provádí přes grid, který umožňuje editovat zobrazená data. Ty jsou následně přenesena do *DataSetu* a uložena.

Následující obrázky 15 a 16 ukazují náhled na dialog a strukturu databáze pro uložení uživatele.



Obrázek 15: Editace uživatele



Obrázek 16: struktura části databáze, nastavení uživatele

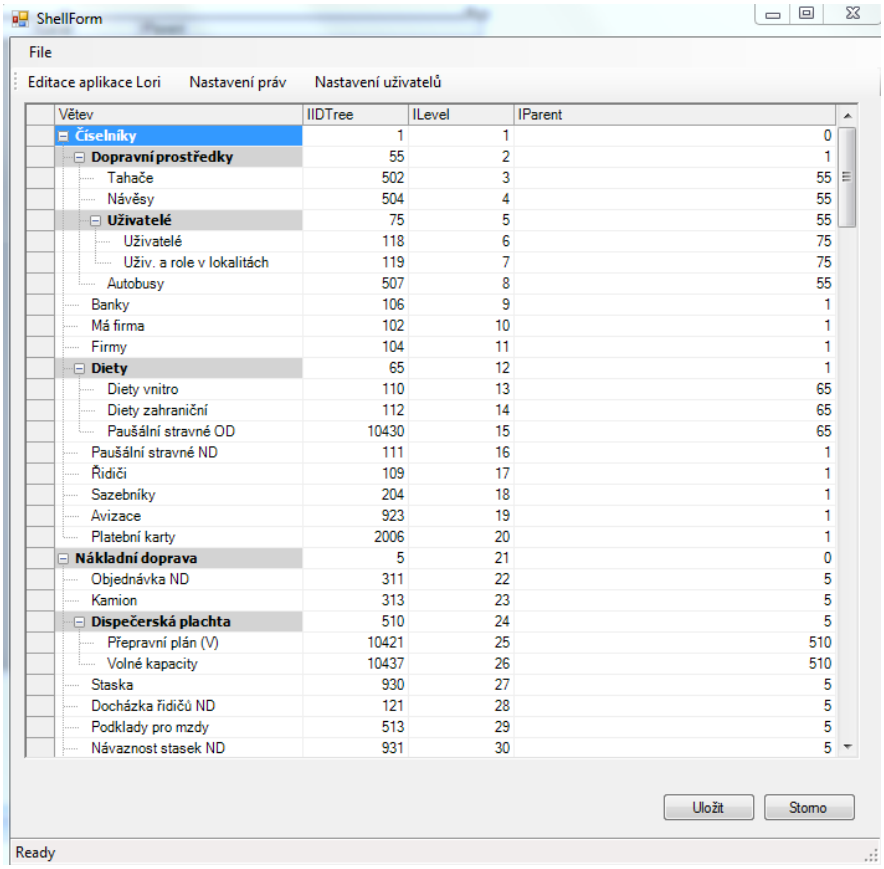
3.2.1.3. Editace modulů

Slouží pro editaci menu v aplikaci *Lori*. Jednotlivé položky je možné přidávat, editovat, mazat a měnit jejich pořadí pouhým táhnutím v mřížce.

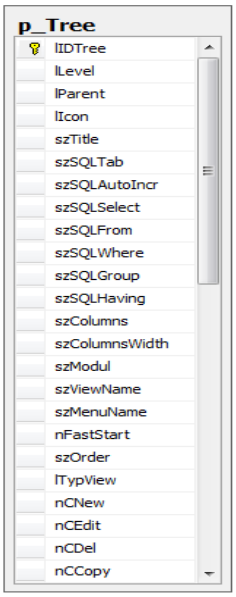
Položka v menu aplikace *Lori* představuje záznam v tabulce *p_tree*. Hierarchie je tvořená pomocí sloupců *lParent*, *lLevel* a primárního klíče tabulky *p_tree lIDTree*. Do sloupce *lParent* se ukládá *lIDTree* nadřazeného řádku. Pořadí položek na stejné úrovni určuje *lLevel*.

Strom aplikace může dosahovat libovolné úrovně. Přesouvání v modulu je řešeno přečíslováním *lParent* a *lLevel* na požadované hodnoty a následným překreslením mřížky. Protože tato operace byla početně náročná a modul měl nepříjemně pomalé odezvy, použil jsem novou technologii *LINQ*, která problém odstranila.

Následující obrázky 17 a 18 ukazují náhled na dialog a strukturu databáze pro editaci modulů.



Obrázek 17: Editace modulů



Obrázek 18: Struktura databáze, editace modulů

4. Závěr

Hlavní náplní této práce bylo popsat základní možnosti technologie SCSF a softwarových továren jako takových.

Myslím, že pro malé projekty je způsob vývoje za pomocí softwarových továren nevhodný. Příliš mnoho času se v těchto případech stráví nad integrací aplikace do rámce vyžadovaného softwarovou továrnou, místo tvoření specifické logiky pro samotnou aplikaci objednanou zákazníkem. S rostoucí složitostí a s tím související potřebou pracovat v týmu, se ale ukazuje větší potenciál této technologie. V případě, že stejnou softwarovou továrnu používáme napříč všemi projekty, je zaučení nového vývojáře o značnou část jednodušší, rychlejší a tím i levnější. U některých velmi složitých aplikací ale můžeme zjistit, že rámec softwarové továrny je až příliš úzký a my ji budeme muset upravit či rozšířit.

Je navíc třeba mít na vědomí, že nad kódem nemáme plnou kontrolu a přesto se na továrnu musíme spolehnout. To sice platí o jakékoliv platformě či frameworku, ale továrny obvykle nemají tak velkou podporu ze strany výrobce jako například samotný .NET Framework. Při výběru softwarové továrny by měla být vždy známa alespoň základní roadmapa, abychom věděli, kam autoři směřují a jestli bude pro továrnu existovat podpora v následujících letech.

Z mých více než ročních zkušeností s vývojem pomocí SCSF mohu konstatovat, že práce na projektech byla rychlejší a efektivnější než běžný způsob vývoje. Jako nevýhodu jsem vnímal nutnost osvojit si značné množství teoretických znalostí a tím způsobenou pomalost vývoje první aplikace.

Podle informací od skupiny *Microsoft patterns & practices* práce na SCSF neustávají. To potvrzuje fakt, že v březnu roku 2010 byla vydaná beta verze a již v květnu byla vydána stabilní verze, která již umožňuje integraci do Visual Studia 2010.

Z mého pohledu jsem tímto zadání bakalářské práce splnil a poskytl základní pohled jak na technologii SCSF, tak na softwarové továrny jako takové. Závěrem tedy nezbývá než konstatovat, že softwarové továrny své opodstanění mají a lze jej pro určité typy projektů bezezbytku doporučit.

5. Literatura

[1] David S. Platt, *Programming Microsoft® Composite UI Application Block and Smart Client Software Factory*, Microsoft Press, 2008. ISBN 9780735624146

[2] Patterns & practices, *Dokumetace SCSF*, 2007. ISBN 80-251-1181-4

[3] Šarmanová Jana, *Teorie zpracování dat*, VŠB , 2003. ISBN 80-248- 0310-0

[4] Šarmanová Jana, *Informační systémy*, VŠB, 2003. ISBN 80-248- 0310-0

[5] Converti Mariano, *Blog* [on-line] V čase použití dostupné na WWW: http://blogs.southworks.net/mconverti/

[6] Nagel Christian, *Programujeme profesionálně*, Computer Press, 2007. ISBN 80-251-1181-4

[7] Dokumentace Smart Client Software Factories (April 2008)

[8] Newman, Rich. I*ntroduction to CAB/SCSF Part 1-25)* [online], V čase použití dostupné na WWW: http://richnewman.wordpress.com/intro-to-cab-toc/

[9] Greer, Derek. *Depency Inversion Principle* [online], V čase použití dostupné na WWW: http://www.aspiringcraftsman.com/tag/dependency-inversionprinciple/

Příloha A – obsah CD

Text bakalářské práce

Zdrojové soubory aplikace

Databáze pro MS SQL 2008